CARNUS
Enseignement supérieur RODEZ

C++

Programmation

Prof : Kamal Boudjelaba

4 novembre 2025

2024–2025

BTS CIEL

Carnus Enseignement Supérieur
Avenue de Saint Pierre - Bourran - 12000 RODEZ, France

IR

Table des matières

Liste des tableaux

1. Installation du logiciel (des outils de base)

Pour programmer en C++, on aura besoin de **trois outils principaux** :

Éditeur de texte (avec coloration syntaxique) : VS Code, Notepad++, etc.
Compilateur (pour traduire le code) : g++, MinGW, Clang, etc.
Débogueur (pour corriger les erreurs) : Souvent inclus dans IDE ou en ligne de commande.

Procédure d'Installation :

Pour simplifier, utiliser un **IDE** (Integrated Development Environment, EDI en français : Environnement de Développement Intégré) qui regroupe ces outils :

- **Code::Blocks :** Gratuit et multiplateforme (Windows, Mac OS 32 bits et Linux).
- **VS Code + extensions C++ :** Gratuit et multiplateforme.
- **XCode :** Gratuit et sur Mac OS seulement.
- **autres :** CLion, Eclipse CDT, etc.

TP : Création d'un programme

1. Lancer votre éditeur ou IDE.
2. Créer un fichier `hello.cpp`.

2. Structure de base d'un programme en C++

Il est très courant d'introduire un langage de programmation en utilisant un programme qui affiche le texte "Hello World" à l'écran. Un exemple simple d'un programme C++ qui fait cela est illustré ci-dessous :

Exemple 2.1

```cpp
#include <iostream>
using namespace std;

int main() {        // Ou int main(int argc, char* argv[])
    cout << "Hello world!" << endl;
    // Ou std::cout << "Hello World!\n"; si on supprime la ligne 2
    return 0;
}
```

Analyse du programme

- `#include <iostream>` : bibliothèque pour entrées/sorties.
- `using namespace std;` : simplifie l'accès aux objets standards.
- `int main() {}` : point d'entrée du programme.
- `cout` : affiche à l'écran.
- `endl` : retour à la ligne.

`iostream` signifie "Input Output Stream" (Flux d'entrée-sortie).
Dans un ordinateur, l'entrée correspond en général au clavier ou à la souris, et la sortie à l'écran.

Remarque 2.1 :

Chaque instruction du code se termine par un point-virgule.

TP : Ecriture, Compilation et Exécution

(a) Copiez le code de l'Exemple 2.1 et enregistrez-le.
(b) Compilez (`g++ hello.cpp -o hello`) et exécutez (`./hello`). Vous pouvez utiliser le menu ou le bouton d'exécution.
(c) Modifiez le message, recompilez, puis lancez-le à nouveau.

NOTE : Erreur classique

Oublier le ; en fin d'instruction : lisez bien les messages du compilateur !

2.1 Les commentaires dans le code

Les commentaires sont nécessaires pour expliquer le fonctionnement du programme. Ils ne sont pas exécutés par le compilateur.

Commentaires sur une ligne :

```
// Votre commentaire
```

Commentaires sur plusieurs lignes :

```
/* Début
du commentaire
multi-lignes */
```

TP : Commenter le code

Ajoutez des commentaires à votre exemple précédent : chaque variable, chaque bloc important.

3. Les variables

Une variable est un espace nommé qui stocke une valeur. Dans les programmes C++, comme dans la plupart des langages compilés, les variables ainsi que leur types doivent être déclarés avant d'être utilisées.

Les règles de nommage :

- Le nom de la variable doit être constitué uniquement de lettres, de chiffres et du tiret-bas "_".
- Le premier caractère doit être une lettre.
- On n'utilise pas d'accents ni d'espaces dans le nom.

Nom du type	Type d'élément de la variable
bool	Une valeur parmi 2 possibilités : vrai (`true`) ou faux (`false`)
char	Un caractère ASCII
int	Un entier
unsigned int	Un nombre entier positif ou nul
double	Un nombre réel
string	Une chaîne de caractères

Table 1. Types de variables

3.1 Déclaration de variables

Une déclaration se fait ainsi : `Type Nom_de_la_variable = Valeur;`

Exemple (dans fonction `main()`) :

```
/* - Déclaration avec initialisation */
int var1 = 16;               // entier valant 16
double var2 = 4.53;          // réel valant 4.53
double erreur = 1.0e-12;     // 10 puissance -12 (notation scientifique)
bool var3 = true;            // booléen vrai
char var4 = 'a';             // caractère 'a'
string nom = "Carnus";       // chaîne de caractères "Carnus"
string chaine = "BTS CIEL"; // chaîne de caractères "BTS CIEL"
/* - Déclaration sans initialisation */
int a;
double x;
bool test;
string phrase;
```

3.2 Les constantes

Une constante est une variable dont la valeur ne peut pas changer :

```
const double pi = 3.14159;
// ou via préprocesseur (moins moderne):
#define PI 3.14159
```

TP : Constantes

Initialiser une variable `const` puis la modifier dans le code. Que vous répond le compilateur ?

3.3 Chaînes de caractères (type `string`)

Pour utiliser `string`, il faut inclure le fichier d'en-tête `string` (`#include <string>`)(mais ici on omet cette ligne).

Exemple d'utilisation :

```
string ville = "Rodez";
cout << "Longueur: " << ville.length() << endl;
cout << "3e lettre: " << ville[2] << endl;
```

Chaîne de caractères

Écrire un programme qui demande une phrase complète, puis affiche sa longueur et la première lettre.

4. Opérateurs logiques et booléens

Opérateur	Symbole	Exemple
ET logique	`&&`	`(a > 0 && b > 0)`
OU logique	`||`	`(a == 0 || b == 0)`
NON logique	`!`	`!(a < 5)`

Table 2. Opérateurs logiques

Retourne un booléen (`true` ou `false`).

```
int x = 5, y = 10;
bool test = (x < y && x != 0);
cout << test << endl; // Affiche 1 (vrai)
```

5. Affichage (cout) et formatage

Pour afficher un texte ou une variable, , on utilise l'instruction `cout <<` (console output) :

```
cout << "Texte " << variable << endl;
```

Caractères spéciaux utiles en sortie :

Commande	Symbole
Nouvelle ligne	\n
Tabulation	\t
Guillemet simple	\'
Guillemet double	\"

Table 3. Commandes de formatage

6. Lecture clavier (cin et getline)

Lecture simple (sans espaces) :

```
int age;
cin >> age;
```

Lecture de ligne complète (avec espaces) :

```
string nomComplet;
getline(cin, nomComplet);
```

Attention : après un `cin >>`, ajoutez `cin.ignore();` avant un `getline()` pour nettoyer le retour chariot restant.

Exemple 6.1: Exemple complet

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    double note;
    cout << "Note sur 20 ? ";
    cin >> note;

    cin.ignore(); // Évite que getline saute la ligne

    string etudiant;
    cout << "Nom complet ? ";
    getline(cin, etudiant);

    cout << etudiant << " a obtenu " << note << "/20." << endl;
    return 0;
}
```

TP : cout et cin

(a) Demander à l'utilisateur son prénom et afficher « Bonjour ... ! » (avec `getline` pour prénoms composés).
(b) Demander l'année de naissance, calculer l'âge (en 2025), vérifier et afficher une erreur avec `cerr` si invalidé (voir ci-dessous).

7. Le flux d'erreurs (cerr)

— `cout` : affichage normal (bufferisé).

— `cerr` : affichage d'erreur (non bufferisé, immédiat).

Exemple :

```
cerr << "Erreur : division par zéro." << endl;
```

Utile pour séparer sorties normales et erreurs.

8. Les opérateurs arithmétiques

Opération	Symbole
Addition	+
Soustraction	−
Multiplication	*
Division	/
Modulo (reste)	%

Table 4. Opérateurs arithmétiques

Exemple division entière *vs* réelle :

```
int a = 7, b = 3;
cout << a / b << endl;          // 2 (division entière)
cout << (double)a / b << endl; // 2.3333 (division réelle)
```

Note 8.1.
L'opérateur % (exp : $a\%b$) renvoie le reste de la division euclidienne de a par b.
L'opérateur modulo est utilisé uniquement sur des entiers.
Dans la division euclidienne de 7 par 3, le quotient est 2 et le reste est 1 ($7 = 2\times3+1$). Donc $7\%3$ renvoie 1.

Remarque 8.1 :

Pour réaliser des calculs mathématiques (racine carrée, sin ...), il faut inclure la librairie `cmath`.
`#include <cmath>`

```
#include <iostream>
#include <cmath>
using namespace std;

int main(int argc, char* argv[])
{
    double x = 1.0, y = 2.0, z;
    z = sqrt(x);    // Racine carrée
    z = exp(y);     // Exponentiel
    z = pow(x, y);  // x à la puissance de y
    z = M_PI;       // Valeur de pi
    return 0;
}
```

Raccourcis des opérateurs

Expression complète	Raccourci
a = a + b;	a += b;
a = a - b;	a −= b;
a = a * b;	a *= b;
a = a / b;	a /= b;
a = a % b;	a %= b;
a = a + 1;	a++;
a = a - 1;	a−−;

Table 5. Raccourcis opérateurs

TP : Les raccourcis

Tester ce programme, puis remplacer la ligne contenant le commentaire 'Opération' par l'une des instructions ci-dessous :

```cpp
#include <iostream>
using namespace std;

int main()
{
   double nombre = 1.5;
   nombre++;         //'Opération'
   cout << "La valeur actuelle est : " << nombre << endl;
   return 0;
}
```

Instructions :

```cpp
++nombre;
nombre--;
--nombre;
nombre += 1.7;
nombre *= 2.;
nombre -= 1.1;
nombre /= 3.;
```

9. Opérateurs bit à bit et décalages

Opérateur	Description	Exemple
`&`	ET bit à bit	`a & b`
`|`	OU bit à bit	`a | b`
^	OU exclusif (XOR)	a ^ b
~	NON bit à bit	$\sim a$
<<	Décalage à gauche	$a << 1$ (mult. par $2^1 = 2$)
>>	Décalage à droite	$a >> 2$ (div. par $2^2 = 4$)

Table 6. Opérateurs bit à bit et décalages

TP : Opérateurs bit à bit

Tester ce programme, puis ajouter des lignes pour afficher les résultats des instructions données ci-dessous :

```cpp
#include <iostream>
using namespace std;

int main()
{
    int a = 5;       // binaire : 0101
    int b = 3;       // binaire : 0011

    cout << (a & b) << endl;  // 1   --> 0001
    return 0;
}
```

Instructions :

```cpp
a | b;
a ^ b;
~a;
a << 1;
a << 3.;
```

10. Exercices

Exercice 1

Écrire un programme qui demande le prénom de l'utilisateur et affiche « Bonjour, ... ! ».

Exercice 2

Lire deux entiers, afficher quotient (division entière), reste, et division réelle (avec conversion).

Exercice 3

Lire un entier, afficher carré et cube (bonus : utiliser `pow()`).

Exercice 4

Lire un entier, un réel, une chaîne (avec espaces), puis afficher.

Exercice 5

Lire un entier, afficher sa valeur multipliée par 2 (`<< 1`) et divisée par 4 (`>> 2`).

11. Structures conditionnelles

11.1 L'instruction `if`

Pour exécuter un bloc d'instructions seulement si une condition est vraie :

```
if (condition) {
    // instructions à exécuter si vrai
}
```

Il est possible de proposer une alternative avec `else` :

```
if (age < 18) {
    cout << "Mineur." << endl;
} else {
    cout << "Majeur." << endl;
}
```

Remarque :

- Les **accolades** sont obligatoires si plusieurs instructions suivent le `if`.
- L'expression entre parenthèses doit renvoyer vrai (`true`) ou faux (`false`).

TP - Structures simples

(a) Écrire un programme qui demande à l'utilisateur de saisir un nombre et qui affiche si ce nombre est positif ou nul (≥ 0), négatif (< 0).
(b) Écrire un programme qui lit l'âge d'une personne et affiche "Mineur" ou "Majeur".

11.2 L'instruction `else if`

Pour enchaîner plusieurs cas distincts :

```
if (note >= 16) {
    cout << "Excellent";
} else if (note >= 12) {
    cout << "Bien";
} else if (note >= 10) {
    cout << "Moyen";
} else {
    cout << "Insuffisant";
}
```

TP - Conditions multiples (Grille de notation)

Écrire un programme qui lit une note sur 20 et affiche une appréciation selon les fourchettes données ci-dessus.

11.3 L'instruction `switch`

Pour gérer de nombreux cas sur une variable entière ou caractère :

```
switch (choix) {
    case 1:
        cout << "Menu 1"; break;
    case 2:
        cout << "Menu 2"; break;
    default:
        cout << "Invalide";
}
```

- Le mot-clé `break` sert à sortir du `switch` après un cas traité.
- Toujours prévoir un cas `default` pour les valeurs inattendues.

TP : Découverte de switch

Écrire un programme qui demande à l'utilisateur de saisir un numéro de mois (1 à 12) et qui affiche le nom du mois correspondant.

TP - Switch

Écrire un programme qui propose un menu simple (1 : Démarrer, 2 : Arrêter, 3 : Redémarrer), lit le choix de l'utilisateur, et affiche l'action correspondante.

11.4 Opérateurs logiques et relationnels

Les opérateurs logiques permettent de combiner plusieurs conditions.
Une premiere utilité de combinaison d'opérateurs logiques et relationnels est de remplacer des instructions if imbriquées par une seule instruction if.

Opérateur	Symbole
AND	&&
OR	||
NOT	!

Table 7. Opérateurs logiques en C++

Relation	Symbole
Égal à	==
Différent de	!=
Supérieur à	>
Supérieur ou égal à	>=
Inférieur à	<
Inférieur ou égal à	<=

Table 8. Opérateurs relationnels en C++

```
double x;
double y;
if ((x > 12) && (x < 14))
{
    y = 10.0; // Les deux conditions sont remplies
}
```

```
double p, q;
int i;
double y;
if ((p > q) || (i != 1))
{
    y = 10.0; // Au moins une des deux est vraie
}
else
{
    y = -10.0; // Aucune des deux n'est vraie
}
```

TP : Plage de valeurs

Écrire un programme qui demande à l'utilisateur un nombre. Indiquer à l'écran s'il est compris entre 10 et 20 inclus.
Essayer de le faire, une fois avec deux tests `if` imbriqués, puis avec une seule condition à l'aide des opérateurs logiques.

TP - Combinaisons de conditions

(a) Lire deux nombres et afficher le plus grand s'ils sont différents.
(b) Si les deux sont positifs, afficher leur somme.

12. Les boucles

12.1 Boucle `for`

Utile quand on connaît le nombre d'itérations :

```
for (initialisation ; condition ; incrementation)
{
    instructions
}
```

La partie en-tête contrôle l'exécution du corps de la boucle.

1. On exécute l'instruction `initialisation`
2. On teste la `condition`
   - si elle est vraie, on exécute `instructions`, puis l'instruction `incrementation` puis on revient au 2
   - si elle est fausse, on passe à la suite du programme (on sort de la boucle `for`).

```
for (int i = 0; i < 5; i++) {
    cout << i << endl;
}
```

TP - Boucle for (affichage)

(a) Afficher les entiers de 1 à 10 en utilisant une boucle `for`.
(b) Afficher les entiers de 10 à 1 dans l'ordre décroissant à l'aide d'une boucle `for`.

12.2 Boucle `while`

Exécuter tant qu'une condition est vraie :

```
while (condition)
{
    Instructions
}
```

1. On teste `condition`
   - si elle est vraie, on exécute `instructions` puis on recommence au 1
   - si elle est fausse, on passe à la suite du programme (on sort de la boucle `while`).

```
int i = 0;
while (i < 5) {
    cout << i << endl;
    i++;
}
```

TP - Boucle while : Saisie contrôlée

(a) Écrire un programme qui demande à l'utilisateur de saisir un nombre **strictement positif** tant que la valeur saisie n'est pas correcte. Afficher le nombre accepté ensuite.

12.3 Boucle `do ... while`

Au moins une exécution, puis on teste :

```
do
{
    Instructions
} while (condition);
```

L'instruction do...while exécute toujours la première itération.

1. On exécute instructions
2. On évalue condition
   - si elle est vraie, on recommence au 1
   - si elle est fausse, on passe à la suite du programme (on sort de la boucle do...while).

```
int val;
do {
    cout << "Tapez 0 pour quitter.";
    cin >> val;
} while (val != 0);
```

TP - do ... while

(a) Écrire un programme qui demande à l'utilisateur de saisir un nombre entre 1 et 10 en utilisant une boucle `do ... while`. Le programme doit continuer à demander un nombre tant que l'utilisateur n'a pas saisi un nombre valide.
(b) Écrire un programme qui lit des entiers jusqu'à ce que l'utilisateur entre un nombre négatif. Afficher la somme de tous les nombres positifs saisis.

TP : Menu interactif

(a) Écrire un programme qui affiche un menu simple avec plusieurs choix (par exemple : 1- Calculer une somme, 2- Convertir une température, 0- Quitter). Le menu doit se répéter jusqu'à ce que l'utilisateur entre 0.

13. Les tableaux

Un **tableau** ("array") regroupe plusieurs éléments de même type qui sont accessibles par leur indice.

Tableau à 1 dimension (vecteur) :

Déclaration : type nom[**taille**]
Cette instruction signifie que le compilateur réserve **taille** places en mémoire pour ranger les éléments du tableau.

— int vecteur[10] : le compilateur réserve des places en mémoire pour 10 entiers
— float nombre[5] : le compilateur réserve des places en mémoire pour 5 réels

Note 13.1.
Un élément du tableau est repéré par son indice. En langage C++ (C et Python) les tableaux commencent à l'indice 0. L'indice maximum est donc **taille** − 1.

```
int notes[5];            // déclare un tableau de 5 entiers
notes[0] = 15;           // affectation au premier élément
float nombre[5]; // 5 décimaux
```

Pour déclarer et initialiser :

```
int valeurs[3] = {10, 5, 7};
double tab[] = {1.2, 3.4, 2.1, 6.0};    // taille automatique
```

Accès à un élément :

```
cout << tab[2] << endl; // Affiche le 3e élément
```

Tableau à 2 dimensions (matrice) :

Déclaration : type nom[**taille1**][**taille2**]

— int matrice[10][3] : tableau de nombres entiers de dimensions 10 lignes et 3 colonnes
— float nombre[2][5] : tableau de nombres réels de dimensions 2 lignes et 5 colonnes

```cpp
int y[2][3] = {{1,4,6},{3,7,5}}; // 2 lignes et 3 colonnes
```

Exemple 13.1: Remplissage manuel et affichage d'une matrice

```cpp
#include <iostream>
using namespace std;

int main() {
    int matrice[2][3]; // Déclaration d'une matrice 2x3

    // Remplissage manuel
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            cout << "Entrez l'élément [" << i << "][" << j << "]: ";
            cin >> matrice[i][j];
        }
    }

    // Affichage
    cout << "Matrice remplie :\n";
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            cout << matrice[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

TP : Manipulation de tableaux 1D

(a) Écrire un programme qui demande à l'utilisateur de saisir 5 nombres dans un tableau. Afficher ensuite toutes les valeurs saisies sur une seule ligne séparées par des espaces.
(Bonus) Afficher la somme des éléments du tableau.

TP - Tableaux 1D

(a) Lire 4 notes et calculer la moyenne.
(b) Afficher la note la plus élevée.

TP - Tableaux 2D

(a) Demander à l'utilisateur de remplir une matrice 3x3.
(b) Calculer la somme des éléments de chaque ligne.

14. Les fonctions

Une **fonction** permet de factoriser le code et d'améliorer la lisibilité et la réutilisation.
Déclaration :

```
type nom_de_la_fonction(arguments)
{
    //Instructions
}
```

- `type` : type du résultat retourné (`int`, `double`, `void`, ...)
- `nom_de_la_fonction` : permet de donner un nom à la fonction
- `arguments` : variables d'entrée (séparées par des virgules)

Exemple :

```
double carre(double x) {
    return x * x;
}
// Appel à la fonction :
double val = carre(3.5); // Renvoie 12.25
```

Paramètres et valeurs de retour

- Une fonction peut recevoir zéro, un ou plusieurs paramètres.
- Elle peut retourner une valeur, ou rien (`void`).

Portée des variables

- Les variables déclarées dans une fonction sont dites **locales** (ne sont visibles que dans cette fonction).
- Une variable globale (déclarée hors de toute fonction) est évitable (déconseillée).

TP : Fonctions de base

(a) Créer une fonction qui calcule le carré d'un entier.

TP : Fonctions personnalisées

(a) Écrire une fonction qui calcule le cube d'un entier.
(b) Réaliser un programme qui lit un entier et affiche son cube en appelant la fonction.

TP - Portée des variables

Écrire un programme avec une fonction qui utilise une variable locale nommée `val`. Ajouter une variable globale aussi nommée `val` et afficher leur valeur respective dans le `main` et dans la fonction.

15. Notions avancées

Introduction aux pointeurs

Un **pointeur** est une variable qui stocke l'adresse mémoire d'une autre variable :

```
int x = 10;
int* ptr = &x;     // ptr contient l'adresse de x
cout << *ptr;      // affiche la valeur pointée, ici 10
```

Utilisation fréquente pour les tableaux, fonctions avancées ou optimisation mémoire.

TP : Pointeurs

(a) Écrire un programme qui lit un entier, puis affiche son double en utilisant un pointeur.
(b) Afficher l'adresse mémoire d'une variable.

Les références &

Permet de passer une variable par référence, donc de la modifier dans une fonction :

```
void incrementer(int& n) {
    n = n + 1;
}
```

TP : Passage par référence

(a) Créer une fonction `ajouter10()` qui ajoute 10 à un entier passé par référence.

Entrées/Sorties de fichiers

Lire/écrire dans un fichier texte nécessite `#include <fstream>` :

```
#include <fstream>
// Ecriture:
ofstream fichier("donnees.txt");
fichier << "Bonjour";
fichier.close();

// Lecture:
ifstream fichIn("donnees.txt");
string ligne;
getline(fichIn, ligne);
cout << ligne << endl;
```

TP : Manipulation de fichiers

1. Écrire dans un fichier texte le nom et le prénom saisis par l'utilisateur.
2. Lire ce fichier texte et afficher son contenu à l'écran.

TP - Fichiers

(a) Écrire un programme qui crée un fichier et y écrit 5 lignes de texte.
(b) Lire le fichier et afficher le contenu à l'écran.

16. Résumé

— Programme bien structuré : initialisation, variables, logique conditionnelle, boucles, fonctions.
— Les variables doivent être typées et initialisées avant usage.
— Utilisez fonctions pour clarifier, commenter votre code.
— Privilégiez la robustesse : Vérifiez les entrées utilisateur et prévoyez des cas d'erreur (division par zéro, mauvaise saisie, etc.).
— Approche progressive : commencez simple, puis ajoutez logique, tableaux, modularisation.